Beeldverwerken Lab 1

Max Bos (10669027) & Philip Bouman (10668667)

April 5, 2017

This document contains answers to the theory questions and notes to the exercises belonging to Lab 1.

2 Interpolation

Theory Answers

2.1 The nearest-neighbour method for a point x is:

$$f(x) = F[\left\lfloor x + \frac{1}{2} \right\rfloor]$$

2.2 The two equations to solve are:

$$a = F(k+1) - F(k)$$
$$b = (1+k)F(k) - kF(k+1)$$

2.3

$$\begin{bmatrix} -1 & 1 \\ 1+k & -k \end{bmatrix} \begin{bmatrix} F(k) \\ F(k+1) \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix}$$

2.4

$$f(x,l) \approx f_1(x,l) = (1-a)F(k,l) + aF(k+1,l)$$
$$(\text{where } a = x - k)$$

The equivalent is:

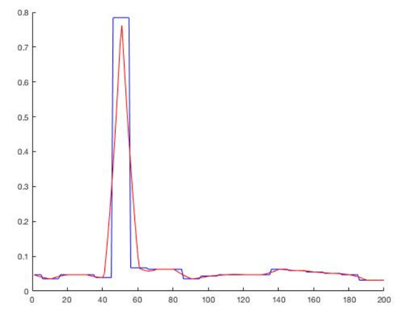
$$f(x,l+1) \approx f1(x,l+1) = (1-a)F(k,l+1) + aF(k+1,l+1)$$

Use lineair interpolation in the orthogonale direction to approximate $f(x,y)$:

$$f(x,y) \approx f_1(x,y) = (1-b)f1(x,l) + bf1(x,l+1)$$
$$(\text{where } b = y - l)$$

Then substitute $f1(x,l)$ and $f1(x,l+1)$ back into the equation:

$$\begin{gathered} f_1(x,y) = (1-a)(1-b)F(k,l)+ \\ (1-a)bF(k,l+1)+ \\ abF(k+1,l+1)+ \\ a(1-b)F(k+1,l) \end{gathered}$$


(a) interpolation methods, blue:nearest / red:linear

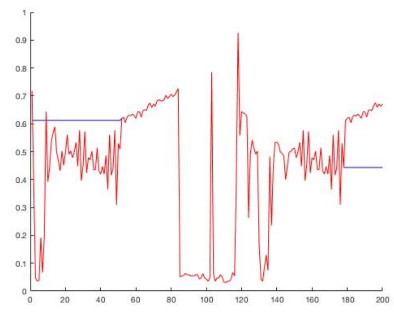

(b) border problem methods, blue:closest / red:periodic

Figure 1: Profiles belonging to different interpolation and border problem methods

Exercise Notes

We've implemented three kinds of methods to solve the border problem. As default, the constant method is implemented which will always return a value of 0 when the provided point is not in image. The second method is the 'nearest' method, which returns the color value from the point that is nearest to the provided point that is not in image. This behaviour can be derived from the profile in Figure 1b, as the value keeps constant when the point is not image. The last method is the 'periodic' method, which repeats the image when the provided point is not in image, this also can be seen from its profile in Figure 1b.

3 Rotation

Theory Answers

3.1 The rotation matrix of a counter clockwise rotation about ϕ radians:

$$R = \begin{bmatrix} cos(\phi) & -sin(\phi) \\ sin(\phi) & cos(\phi) \end{bmatrix}$$

3.2 You can transform a point $(x, y)^T$ to a rotated point $(x', y')^T$ around the origin, by multiplying the rotation matrix R with $(x, y)^T$:

$$\begin{bmatrix} cos(\phi) & -sin(\phi) \\ sin(\phi) & cos(\phi) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

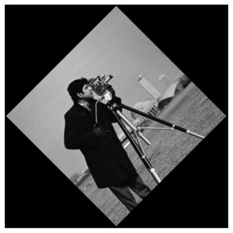
3.3 If you want to rotate a point $\mathbf{x}$ about an arbitrary point $\mathbf{c}$ instead of the origin. First subtract point $\mathbf{c}$ from $\mathbf{x}$ which moves the point to the origin (as the rotation matrix will perform a rotation about the origin). Then, perform the rotation on this new point, and lastly move the point back by adding point $\mathbf{c}$:

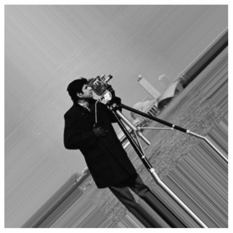
$$\mathbf{x} \mapsto R_\phi(\mathbf{x} - \mathbf{c}) + \mathbf{c}$$

3.4 Nothing will happen. When it is moved back to the origin, the rotation will have no affect. After that it is moved back so simply it is just the point:

$$\begin{bmatrix} cos(\phi) & -sin(\phi) \\ sin(\phi) & cos(\phi) \end{bmatrix} \begin{bmatrix} c_1 - c_1 \\ c_2 - c_2 \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

Exercise Notes


(a) image after rotation w/ constant border method


(b) image after rotation w/ periodic border method

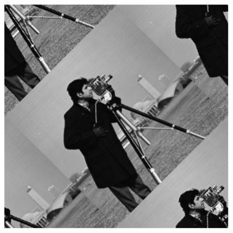

(c) image after rotation w/ periodic border method

Figure 2: Rotating image with an angle of 45 degree without loss

We've implemented two means for performing a rotation on an image which result in a complete picture: an integrated form into rotateImage() and one that uses borderImage() before calling rotateImage(). One would use borderImage() by first adding a border around an image using borderImage() and then provide this bordered image as input to rotateImage(), border methods would have no result on this approach as the coordinates specified into pixelValue() will always fall inside the image domain. The integrated form in rotateImage() does allow the border method to add the correct pixel values when a coordinate falls outside the domain of the original image, see Figure 2.

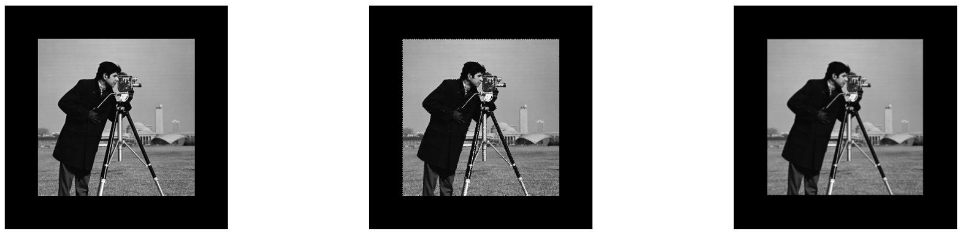
The intelligent border is achieved by calculating the dimension (width and height) of the image after the rotation of a specified angle is performed.

```
newWidth = abs(height * cos(angle) + width * sin(angle));
newHeight = abs(height * sin(angle) + width * cos(angle));
topMargin = ceil((newWidth - width) / 2);
leftMargin = ceil((newHeight - height) / 2);
newWidth = width + leftMargin*2;
newHeight = height + topMargin*2;
```

The difference between the new dimension and original dimension divided by 2, results in the margins needed to position the original image in the center of its new domain.

To compare the quality of both interpolation methods mathematically, we've implemented a distance measure in a quadratic manner using Euclidean distance. First, the Euclidean distances between each pair of observations in the original image data matrix X and the resulting image matrix Y is calculated (using Matlab `pdist2`), over which the square root sum is calculated.

```
D = pdist2(X, Y, 'euclidean');
distance = sqrt(sum(D(:)));
```



(a) original image with intelligent border (b) after double rotation w/ nearest interpolation (c) after double rotation w/ linear interpolation

Figure 3: Double rotation applied on a bordered image

The results are of the following:

```
    Distance (nearest): 871.617152
    Distance (linear): 867.616492
    Distance difference: 4.000659
```

It seems that using (bi)linear interpolation yields a lower distance. At first glance, the resulting images after nearest and linear interpolation look similar to the original image. The quality of the linear interpolation method seems smoother than the quality of nearest. The black border around the picture does seem to have an influence on the distance measure, as there seem to be missing pixels around the border of the resulting picture for nearest neighbour interpolation.

4 Affine Transformations

Theory Answers

4.1 A problem that we run into is that a 2 x 3 matrix cannot be multiplied by a 2 x 1 matrix. Use of homogeneous coordinates is needed to write the given equations as a matrix equation.

4.2 With the use of homogeneous coordinates, translation combined with linear transformations can be expressed using matrix multiplication. The given equations can be written as the following matrix multiplication, using homogeneous coordinates:

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

4.3

$$\begin{bmatrix} cos(\phi) & -sin(\phi) & c \\ sin(\phi) & cos(\phi) & f \end{bmatrix}$$

The above matrix represents an affine transformation that is capable of a counter-clockwise rotation of angle ϕ combined with a translation with vector $\begin{bmatrix} c \\ f \end{bmatrix}$.

4.4 An easy way to find three point-pairs is by pairing three corner points of the original image to their mapped corner point in the output image. The output image has four corner points available, so the ouput points can be three of $(1,1)^T$, $(1,N)^T$, $(M,1)^T$ and $(M,N)^T$, where M is the number of rows and N is the number of columns.

4.5 We need three points, namely one point vector and two direction vectors, because this allows calculation of the angle of transformation. Using these three vectors we can derive the fourth point, as the output image is a parallelogram. If we use the three input points: $(1,1)^T$, $(1,N)^T$, $(M,1)^T$. The other corner point of the square is reached as $(M,N)^T = (1,1)^T + (1,N)^T + (M,1)^T$, and is therefore fully determined by the linearity properties of the transformation: it is at the sum of the transformed direction vectors. We do not use more than three points, as three is sufficient to calculate the corresponding transformation matrix.

Exercise Notes

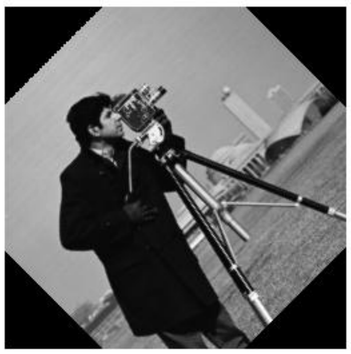


Figure 4: Image after affine transformation

Please see Figure 4 for the end result of the affine transformation, to perform a counter-clockwise rotation of 45 degrees. The parameters used to call myAffine() are the following:

```
myAffine(img, -54, 128, 128, -54, 128, 310, 256, 256, 'linear')
```

5 Re-projecting Images

Theory Answers

5.1

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11}x + m_{12}y + m_{13} \\ m_{21}x + m_{22}y + m_{23} \\ m_{31}x + m_{32}y + m_{33} \end{bmatrix} = \begin{bmatrix} \lambda x' \\ \lambda y' \\ \lambda \end{bmatrix}$$

5.2
$$\begin{bmatrix} m_{11}x + m_{12}y + m_{13} \\ m_{21}x + m_{22}y + m_{23} \\ m_{31}x + m_{32}y + m_{33} \end{bmatrix} = \begin{bmatrix} \lambda x' \\ \lambda y' \\ \lambda \end{bmatrix}$$

We're not interested in λ, so λ is eliminated. Thus, a point correspondence gives 2 equations, namely:

$$m_{11}x + m_{12}y + m_{13} = m_{31}xx' + m_{32}yx' + m_{33}x'$$
$$m_{21}x + m_{22}y + m_{23} = m_{31}xy' + m_{32}yy' + m_{33}y'$$

5.3 the projective transformation can transform any arbitrary quadrilateral to another arbitrary quadrilateral. So, the projective transformation should at least have 16 parameters, as we need to map 4 input points to 4 output points, where each point consists of 2 parameters.

5.4 We can normalize the matrix M, by setting $m_{33} = 1$, because m_{33} is not relevant to the projective transformation. This leaves us with 8 relevant parameters in matrix M.

5.5 We therefore need a minimum of 4 point correspondences to determine a projective transformation in 2D. (Bonus:) And a minimum of 8 point correspondences in 3D.

5.6 The unknowns of the transformation, collected in a vector, are:

$$\vec{m} = \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{31} \\ m_{32} \\ m_{33} \end{bmatrix}$$

5.7 All n equations (for $n/2$ point correspondences) rewritten into a matrix equation:

$$\begin{bmatrix} u_1 & v_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -x_1v_1 & -x_1 \\ 0 & 0 & 0 & u_1 & v_1 & 1 & -y_1u_1 & -y_1v_1 & -y_1 \\ u_2 & v_2 & 1 & 0 & 0 & 0 & -x_2u_2 & -x_2v_2 & -x_2 \\ 0 & 0 & 0 & u_2 & v_2 & 1 & -y_2u_2 & -y_2v_2 & -y_2 \\ . & . & . & . & . & . & . & . & . \\ u_n & v_n & 1 & 0 & 0 & 0 & -x_nu_n & -x_nv_n & -x_n \\ 0 & 0 & 0 & u_n & v_n & 1 & -y_nu_n & -y_nv_n & -y_n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{31} \\ m_{32} \\ m_{33} \end{bmatrix} = \vec{0}$$

5.8 You can avoid this by calculating $\vec{x}$ to be in the kernel (nullspace) of matrix A:

$$\vec{x} = ker(A)$$

5.9 Because we consider a 2D projective transformation in this case. A 2D projective transformation doesn't depend on its last coefficient (m_{33}).

5.10 Because A is an 8 x 9 matrix with general coefficients, the kernel is 1-dimensional. So we can use Matlab's `null(A)` command to calculate $\vec{x}$.

Exercise Notes

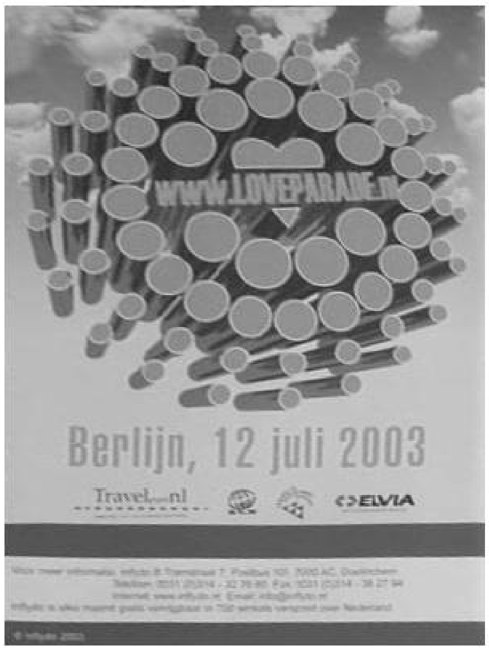


Figure 5: Straightened flyer as result of a projective transformation on flyers.png

7 Estimating a Camera's Projection Matrix

Theory Answers

7.1

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} = \begin{bmatrix} \lambda_i x_i \\ \lambda_i y_i \\ \lambda_i \end{bmatrix}$$

Solving the above equation results to the following three equations:

$$\begin{aligned} m_{11}X_i + m_{12}Y_i + m_{13}Z_i + m_{14} = \lambda_i x_i \\ m_{21}X_i + m_{22}Y_i + m_{23}Z_i + m_{24} = \lambda_i y_i \\ m_{31}X_i + m_{32}Y_i + m_{33}Z_i + m_{34} = \lambda_i \end{aligned}$$

λ can be eliminated, this results in the following two equations:

$$\begin{aligned} m_{11}X_i + m_{12}Y_i + m_{13}Z_i + m_{14} = m_{31}X_ix_i + m_{32}Y_ix_i + m_{33}Z_ix_i + m_{34}x_i \\ m_{21}X_i + m_{22}Y_i + m_{23}Z_i + m_{24} = m_{31}X_iy_i + m_{32}Y_iy_i + m_{33}Z_iy_i + m_{34}y_i \end{aligned}$$

Setting the equations to zero:

$$\begin{aligned} m_{11}X_i + m_{12}Y_i + m_{13}Z_i + m_{14} - m_{31}X_ix_i - m_{32}Y_ix_i - m_{33}Z_ix_i - m_{34}x_i = 0 \\ m_{21}X_i + m_{22}Y_i + m_{23}Z_i + m_{24} - m_{31}X_iy_i - m_{32}Y_iy_i - m_{33}Z_iy_i - m_{34}y_i = 0 \end{aligned}$$

The above two equations can be written as a matrix equation capturing n number of point correspondences:

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -X_1x_1 & -Y_1x_1 & -Z_1x_1 & -x_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -X_1y_1 & -Y_1y_1 & -Z_1y_1 & -y_1 \\ . & . & . & . & . & . & . & . & . & . & . & . \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -X_nx_n & -Y_nx_n & -Z_nx_n & -x_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -X_1y_n & -Y_ny_n & -Z_ny_n & -y_n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{bmatrix} = \vec{0}$$

As can be concluded from the above equation:

$$A = \begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -X_1x_1 & -Y_1x_1 & -Z_1x_1 & -x_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -X_1y_1 & -Y_1y_1 & -Z_1y_1 & -y_1 \\ . & . & . & . & . & . & . & . & . & . & . & . \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -X_nx_n & -Y_nx_n & -Z_nx_n & -x_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -X_1y_n & -Y_ny_n & -Z_ny_n & -y_n \end{bmatrix}$$

7.2 Matrix A will still represent the resulting equations if you have many more point correspondences. As $A\mathbf{m} = \mathbf{0}$ is a matrix multiplication where an $M\times12$ matrix is multiplied with a 12×1 vector. Adding more point correspondences will only increase M, which will always result in an $M\times1$ vector. Adding more points correspondences helps in yielding a more accurate estimation of transformation matrix M.

7.3 No, there will not be an exact solution to $A\mathbf{m} = \mathbf{0}$, as A has no non-trivial nullspace.

7.4 Matrix A has no non-trivial nullspace, so the best solution $\vec{m}*$ is found by calculating the singular value decomposition of A and selecting the last column of this SVD as $\vec{m}*$. This is the best solution since it corresponds to the smallest singular value of A, and thus approaches the nullspace of A.

Projecting Cubes into a Picture (Pseudo 3D)

Exercise Notes

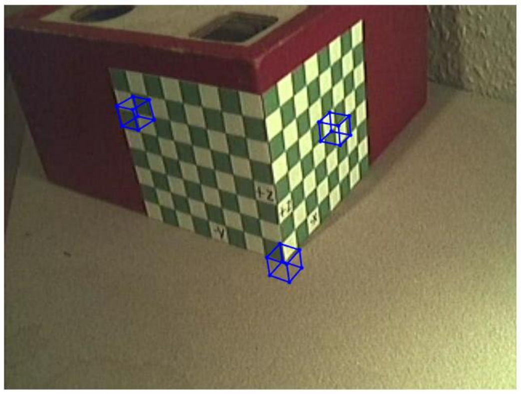
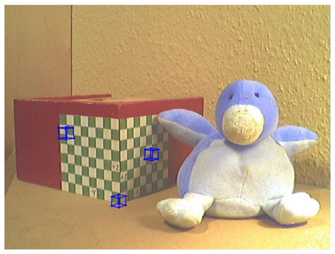
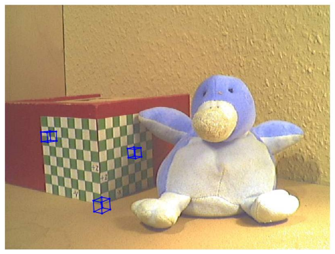


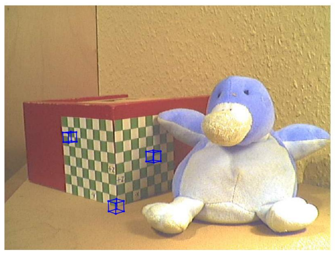
Figure 6: callibrationpoints.jpg with cubes



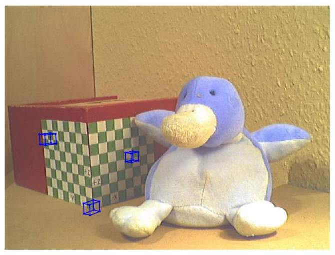
(a) View 1



(b) View 2



(c) View 3



(d) View 4

Figure 7: 4 different views with cubes